

Understanding XML and Its Impact on the Enterprise

"By 2003, more than 95% of the G2000 organizations will deploy XML-based content management infrastructures."

META Group (2000)

In this chapter you will learn:

- Why XML is the cornerstone of the Semantic Web
- Why XML has achieved widespread adoption and continues to expand to new areas of information processing
- How XML works and the mechanics of related standards like namespaces and XML Schema

Once you understand the core concepts, we move on to examine the impact of XML on the enterprise. Lastly, we examine why XML itself is not enough and the current state of confusion as different technologies compete to fill in the gaps.

Why Is XML a Success?

XML has passed from the early-adopter phase to mainstream acceptance. Currently, the primary use of XML is for data exchange between internal and external organizations. In this regard, XML plays the role of interoperability mechanism. As XQuery and XML Schema (see sidebar) achieve greater maturity and adoption, XML may become the primary syntax for all enterprise

data. Why is XML so successful? XML has four primary accomplishments, which we discuss in detail in the sections that follow:

- XML creates application-independent documents and data.
- It has a standard syntax for meta data.
- It has a standard structure for both documents and data.
- XML is not a new technology (not a 1.0 release).

A key variable in XML's adoption, one that possibly holds even more weight than the preceding four accomplishments, is that computers are now fast enough and storage cheap enough to afford the luxury of XML. Simply put, we've been dancing around the concepts in XML for 20 years, and it is only catching fire now because computers are fast enough to handle it. In this regard, XML is similar to the rise of virtual machine environments like .NET and Java. Both of these phenomena would simply have been rejected as too slow five years ago. The concepts were known back then, but the technology was just not practical. And this same logic applies to XML.

Now let's examine the other reasons for XML's success. XML is application-independent because it is plaintext in human-readable form. Figure 3.1 shows a simple one-line word-processing document. Figure 3.2 and Listing 3.1 contrast XML to a proprietary binary format like Microsoft Word for the one-line document shown in Figure 3.1. In contrast, Figure 3.2 is a string of binary numbers (shown in base 16, or hexadecimal, format) where only the creators of the format understand it (some companies attempt to reverse-engineer these files by looking for patterns). Binary formats lock you into applications for the life of your data. Encoding XML as text allows any program to open and read the file. Listing 3.1 is plaintext, and its intent is easily understood.

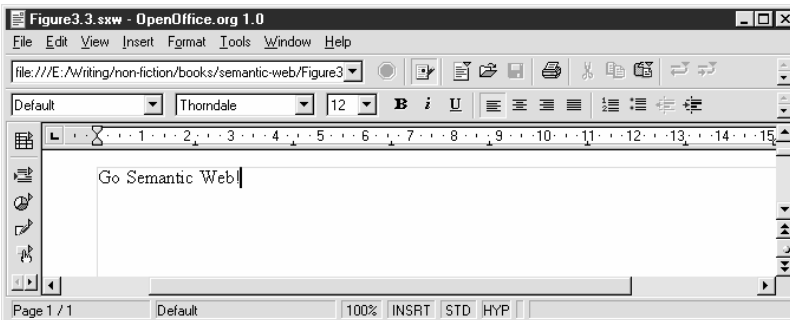


Figure 3.1 A one-line document in a word processor (Open Office).

XQuery and XML Schema in a Nutshell

XQuery is an XML-based query language for querying XML documents. A query is a search statement to retrieve specific portions of a document that conform to a specified search criterion. XQuery is defined further in Chapter 6. XML Schema is a markup definition language that defines the legal names for elements and attributes, and the legal hierarchical structure of the document. XML Schema is discussed in detail later in this chapter.

By using an open, standard syntax and verbose descriptions of the meaning of data, XML is readable and understandable by everyone—not just the application and person that produced it. This is a critical underpinning of the Semantic Web, because you cannot predict the variety of software agents and systems that will need to consume data on the World Wide Web. An additional benefit for storing data in XML, rather than binary data, is that it can be searched as easily as Web pages.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE office:document-content PUBLIC "-//OpenOffice.org//DTD Office-
Document 1.0//EN" "office.dtd"><office:document-content
xmlns:office="http://openoffice.org/2000/office"
...
xmlns:script="http://openoffice.org/2000/script" office:class="text"
office:version="1.0">
<office:script/>
<office:font-decls>
...
<style:font-decl style:name="Thorndale" fo:font-family="Thorndale"
style:font-family-generic="roman" style:font-pitch="variable"/>
</office:font-decls>
<office:automatic-styles/>
<office:body>
<text:sequence-decls>
...
</text:sequence-decls>
<text:p text:style-name="Standard">Go Semantic Web!</text:p>
</office:body>
</office:document-content>
```

Listing 3.1 XML format of Figure 3.1 (portions omitted for brevity).

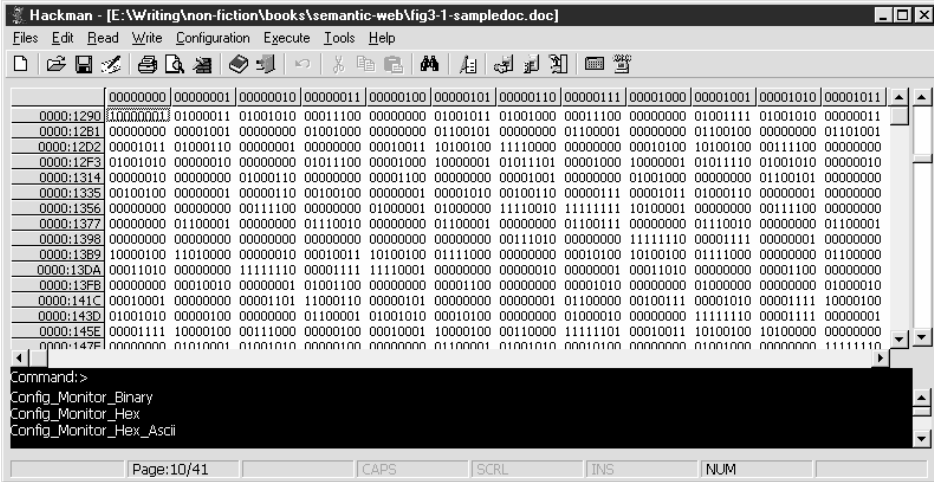


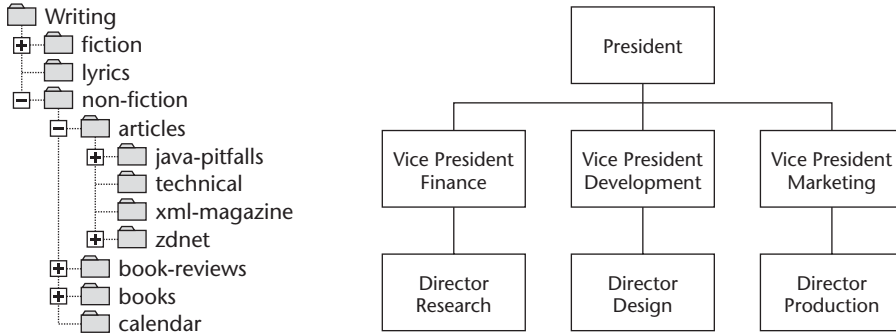
Figure 3.2 Binary MS Word format of the same one line in Figure 3.1 (portions omitted for brevity).

The second key accomplishment is that XML provides a simple, standard syntax for encoding the meaning of data values, or *meta data*. An often-used definition of meta data is “data about data.” We discuss the details of the XML syntax later. For now what is important is that XML standardizes a simple, text-based method for encoding meta data. In other words, XML provides a simple yet robust mechanism for encoding semantic information, or the meaning of data. Table 3.1 demonstrates the difference between meta data and data. It should be evident that the data is the raw context-specific values and the meta data denotes the meaning or purpose of those values.

The third major accomplishment of XML is standardizing a structure suitable to express semantic information for both documents and data fields (see the sidebar comparing them). The structure XML uses is a hierarchy or tree structure. A good common example of a tree structure is an individual’s filesystem on a computer, as shown in Figure 3.3. The hierarchical structure allows the user to decompose a concept into its component parts in a recursive manner.

Table 3.1 Comparing Data to Meta Data

DATA	META DATA
Joe Smith	Name
222 Happy Lane	Address
Sierra Vista	City
AZ	State
85635	Zip code



Folders

Organization Chart

Figure 3.3 Sample trees as organization structures.

The last accomplishment of XML is that it is not a new technology. XML is a subset of the Standardized Generalized Markup Language (SGML) that was invented in 1969 by Dr. Charles Goldfarb, Ed Mosher, and Ray Lorie. So, the concepts for XML were devised over 30 years ago and continuously perfected, tested, and broadly implemented. In a nutshell, XML is “SGML for the Web.” So, it should be clear that XML possesses some compelling and simple value propositions that continue to drive its adoption. Let’s now examine the mechanics of those accomplishments.

The Difference between Documents and Data Fields

An electronic document is the electronic counterpart of a paper document. As such, it is a combination of both content (raw information) and presentation instructions. Its content uses natural language in the form of sentences, paragraphs, and pages. In contrast, data fields are atomic name/value pairs processable by a computer and are often captured in forms.

Both types of information are widespread in organizations, and both have strengths and weaknesses. A significant strength of XML is that it enables meta data attachment (markup) on both of these data sources. Thus XML, bridges the gap between documents and data to enable them to both participate in a single web of information.

What Is XML?

XML is not a language; it is actually a set of syntax rules for creating semantically rich markup languages in a particular domain. In other words, you *apply* XML to create new languages. Any language created via the rules of XML, like the Math Markup Language (MathML), is called an application of XML. A markup language's primary concern is how to add semantic information about the raw content in a document; thus, the vocabulary of a markup language is the external "marks" to be attached or embedded in a document. This concept of adding marks, or semantic instructions, to a document has been done manually in the text publishing industry for years. Figure 3.4 shows the manual markup for page layout of a school newspaper.

As publishing moved to electronic media, several languages were devised to capture these marks alongside content like TeX and PostScript (see Listing 3.2).

```
\documentstyle[doublespace,12pt]{article}
\title{An Example of Computerized Text Processing}
\author{A. Student}
\date{8 June 1993}
\begin{document}
\maketitle
```

```
This is the text of your article. You can
type in the material without being
concerned about ends of lines and word
spacing. LaTeX will handle the spacing for
you.
```

```
The default type size is 10 point.
```

```
The Roman type font is used. Text is
justified and double spaced. Paragraphs are
separated by a blank line.
```

```
\end{document}
```

Listing 3.2 Markup in TeX.

MAXIM

Markup is separate from content.

So, the first key principle of XML is *markup is separate from content*. A corollary to that principle is that markup can surround or contain content. Thus, a

markup language is a set of words, or marks, that surround, or “tag,” a portion of a document’s content in order to attach additional meaning to the tagged content. The mechanism invented to mark content was to enclose each word of the language’s vocabulary in a less-than sign (<) and a greater-than sign (>) like this:

<auto>

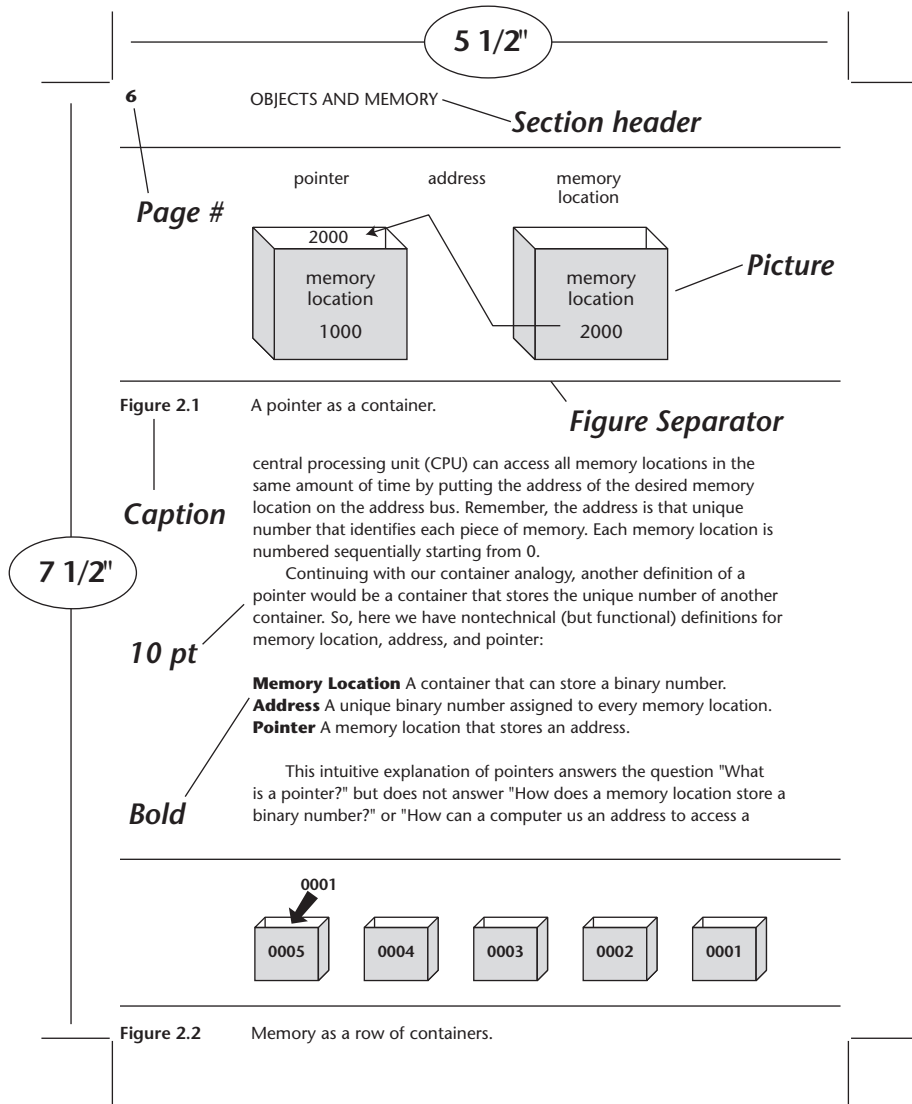


Figure 3.4 Manual markup on a page layout.

To use the < and > characters as part of a tag, these characters cannot be used in content, and therefore they are replaced by the special codes (called entities) > (for greater than) and < (for less than). This satisfies our requirement to separate a mark from content but does not yet allow us to surround, or contain, content. Containing content is achieved by wrapping the target content with a start and end tag. Thus, each vocabulary word in our markup language can be expressed in one of three ways: a start tag, an end tag, or an empty tag. Table 3.2 demonstrates all three tag types.

The start and end tags are used to demarcate the start and end of the tagged content, respectively. The empty tag is used to embed semantic information that does not surround content. A good example of the use of an empty tag is the image tag in HTML, which looks like this: . An image tag does not need to surround content, as its purpose is to insert an image at the place where the tag is. In other words, its purpose is to be embedded at a specific point in raw content and not to surround content. Thus, we can now extend our first principle of XML to this: *Markup is separate from content and may contain content.*

MAXIM

Markup is separate from content and may contain content.

We can now formally introduce an XML definition for the term XML *element*. An XML element is an XML container consisting of a start tag, content (contained character data, subelements, or both), and an end tag—except for empty elements, which use a single tag denoting both the start and end of the element. The content of an element can be other elements. Following is an example of an element:

```
<footnote>
  <author> Michael C. Daconta </author>, <title> Java Pitfalls </title>
</footnote>
```

Here we have one element, called “footnote,” which contains character data and two subelements: “author” and “title.”

Table 3.2 Three Types of XML Tags

TAG TYPE	EXAMPLE
Start tag	<author>
End tag	</author>
Empty tag	

Another effect of tagging content is that it divides the document into semantic parts. For example, we could divide this chapter into <chapter>, <section>, and <para> elements. The creation of diverse parts of a whole entity enables us to classify, or group, parts, and thus treat them differently based on their membership in a group. In XML, such classification begins by constraining a valid document to be composed of a single element, called the *root*. In turn, that element may contain other elements or content. Thus, we create a hierarchy, or tree structure, for every XML document. Here is an example of the hierarchy of an XHTML document (see sidebar on XHTML):

```
<html>
  <head>
    <title> My web page </title>
  </head>
  <body>
    Go Semantic Web!!
  </body>
</html>
```

Listing 3.3 A single HTML root element.

The second key principle of XML is this: *A document is classified as a member of a type by dividing its parts, or elements, into a hierarchical structure known as a tree.* In Listing 3.3, an HTML document starts with a root element, called “html.” which contains a “head” element and a “body” element. The head and body element can contain other subelements and content as specified by the HTML specification. Thus, another function of XML is to classify all parts of a document into a single hierarchical set of parts.

MAXIM

An XML document is classified as a member of a type by dividing its parts, or elements, into a hierarchical structure known as a tree.

XHTML in a Nutshell

XHTML is a reformulation of HTML as an XML application. In practical terms, this boils down to eliminating the laxness in HTML by requiring things like strict nesting, corresponding end tags with all nonempty elements, and adding the forward slash to the empty element tag.

In the discussion about empty elements, we used the `` tag, which included a name/value pair (`src = "apple.gif"`). Each name/value pair attached to an element is called an *attribute*. Attributes only appear in start tags or empty element tags. It has a name (`src`) followed by an equal sign, followed by a value surrounded in either single or double quotes. An element may have more than one attribute. Here is an example of an element with three attributes:

```
<auto color="red" make="Dodge" model="Viper"> My car </auto>
```

The combination of elements and attributes makes XML well suited to model both relational and object-oriented data. Table 3.3 shows how attributes and elements correlate to the relational and object-oriented data models.

Overall, XML's information representation facilities of elements, attributes, and a single document root implement the accomplishments outlined in the first section in the chapter.

Why Should Documents Be Well-Formed and Valid?

The XML specification defined two levels of conformance for XML documents: *well-formed* and *valid*. Well-formedness is mandatory, while validity is optional. A well-formed XML document complies with all the W3C syntax rules of XML (explicitly called out in the XML specification as well-formedness constraints) like naming, nesting, and attribute quoting. This requirement guarantees that an XML processor can parse (break into identifiable components) the document without error. If a compliant XML processor encounters a well-formedness violation, the specification requires it to stop processing the document and report a fatal error to the calling application.

A valid XML document references and satisfies a schema. A *schema* is a separate document whose purpose is to define the legal elements, attributes, and structure of an XML instance document. In general, think of a schema as defining the legal vocabulary, number, and placement of elements and attributes in your markup language. Therefore, a schema defines a particular type or class of documents. The markup language constrains the information to be of a certain type to be considered "legal." We discuss schemas in more detail in the next section.

Table 3.3 Data Modeling Similarities

XML	OO	RELATIONAL
Element	Class	Entity
Attribute	Data member	Relation

W3C-compliant XML processors check for well-formedness but may not check for validity. Validation is often a feature that can be turned on or off in an XML parser. Validation is time-consuming and not always necessary. It is generally best to perform validation either as part of document creation or immediately after creation.

What Is XML Schema?

XML Schema is a definition language that enables you to constrain conforming XML documents to a specific vocabulary and a specific hierarchical structure. The things you want to define in your language are element types, attribute types, and the composition of both into composite types (called complex types). XML Schema is analogous to a database schema, which defines the column names and data types in database tables. XML Schema became a W3C Recommendation (synonymous with standard) on May 5, 2001. XML Schema is not the only definition language, and you may hear about others like Document Type Definitions (DTDs), RELAX NG, and Schematron (see the sidebar titled “Other Schema Languages”).

As shown in Figure 3.5, we have two types of documents: a schema document (or definition document) and multiple instance documents that conform to the schema. A good analogy to remember the difference between these two types of documents is that a *schema* definition is a blueprint (or template) of a type and each *instance* is an incarnation of that template. This also demonstrates the two roles that a schema can play:

- Template for a form generator to generate instances of a document type
- Validator to ensure the accuracy of documents

Both the schema document and the instance document use XML syntax (tags, elements, and attributes). This was one of the primary motivating factors to replace DTDs, which did not use XML syntax. Having a single syntax for both definition and instance documents enables a single parser to be used for both.

Referring back to our database analogy, the database schema defines the columns, and the table rows are instances of each definition. Each instance document must “declare” which definition document (or schema) it adheres to. This is done with a special attribute attached to the root element called “`xsi:noNamespaceSchemaLocation`” or “`xsi:schemaLocation`.” The attribute used depends on whether your vocabulary is defined in the context of a namespace (discussed later in this chapter).

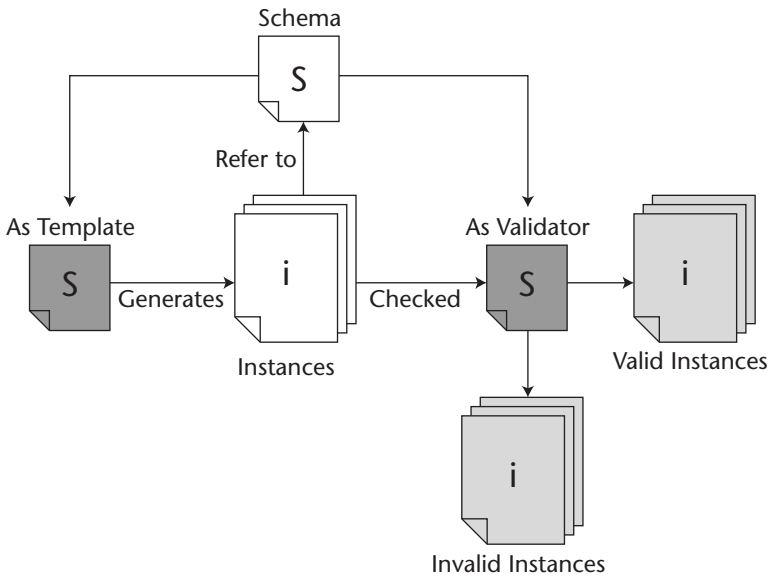


Figure 3.5 Schema and instances.

XML Schemas allow validation of instances to ensure the accuracy of field values and document structure at the time of creation. The accuracy of fields is checked against the type of the field; for example, a quantity typed as an integer or money typed as a decimal. The structure of a document is checked for things like legal element and attribute names, correct number of children, and required attributes. All XML documents should be checked for validity before they are transferred to another partner or system.

What Do Schemas Look Like?

An XML Schema uses XML syntax to declare a set of simple and complex type declarations. A *type* is a named template that can hold one or more values. Simple types hold one value. Complex types are composed of multiple simple types. So, a type has two key characteristics: a name and a legal set of values. Let's look at examples of both simple and complex types.

A simple type is an element declaration that includes its name and value constraints. Here is an example of an element called "author" that can contain any number of text characters:

```
<xsd:element name="author" type="xsd:string" />
```

The preceding element declaration enables an instance document to have an element like this:

```
<author> Mike Daconta </author>
```

Other Schema Languages

While there are dozens of schema languages (as this is a popular topic for experimentation), we will discuss the top three alternatives to XML Schema: DTD, RELAX NG, and Schematron.

Document Type Definition (DTD) was the original schema definition language inherited from SGML, and its syntax is defined as part of the XML 1.0 Recommendation released on February 10, 1998. Some markup languages are still defined with DTDs today, but the majority of organizations have switched or are considering switching to XML Schema. The chief deficiencies of DTDs are their non-XML syntax, their lack of data types, and their lack of support for namespaces. These were the top three items XML Schema set out to fix.

RELAX NG is the top competitor to the W3C's XML Schema and is considered technically superior to XML Schema by many people in the XML community. On the other hand, major software vendors like Microsoft and IBM have come out strongly in favor of standardizing on XML Schema and fixing any deficiencies it has. RELAX NG represents a combination of two previous efforts: RELAX and TREX. Here is the definition of RELAX NG from its specification: "A RELAX NG schema specifies a pattern for the structure and content of an XML document. A RELAX NG schema thus identifies a class of XML documents consisting of those documents that match the pattern. A RELAX NG schema is itself an XML document."¹ For interoperability, RELAX NG can use the W3C XML Schema data types.

Schematron is an open source XML validation tool that uses a combination of patterns, rules, and assertions made up of XPath expressions (see Chapter 6 for a discussion of XPath) to validate XML instances. It is interesting to note that rule-based validation is a different approach from the more common, grammar-based approach used in both XML Schema and RELAX NG.

Sun Microsystems, Inc. offers a free Java-based tool called the Multi-Schema Validator. This tool validates RELAX NG, RELAX Namespace, RELAX Core, TREX, XML DTDs, and a subset of XML Schema.

Notice that the type attributed in the element declaration declares the type to be "xsd:string". A *string* is a sequence of characters. There are many built-in data types defined in the XML Schema specification. Table 3.4 lists the most common. If a built-in data type does not constrain the values the way the document designer wants, XML Schema allows the definition of custom data types.

¹James C. Clark and Murata Makoto, "RELAX NG Tutorial," December 3, 2001. Available at <http://www.oasis-open.org/committees/relax-ng/tutorial.html>.

Table 3.4 Common XML Schema Primitive Data Types

DATA TYPE	DESCRIPTION
string	Unicode characters of some specified length.
boolean	A binary state value of true or false.
ID	A unique identifier attribute type from the 1.0 XML Specification.
IDREF	A reference to an ID.
integer	The set of whole numbers.
long	long is derived from integer by fixing the values of maxInclusive to be 9223372036854775807 and minInclusive to be -9223372036854775808.
int	int is derived from long by fixing the values of maxInclusive to be 2147483647 and minInclusive to be -2147483648.
short	short is derived from int by fixing the values of maxInclusive to be 32767 and minInclusive to be -32768.
decimal	Represents arbitrary precision decimal numbers with an integer part and a fraction part.
float	IEEE single precision 32-bit floating-point number.
double	IEEE double-precision 64-bit floating-point number.
date	Date as a string defined in ISO 8601.
time	Time as a string defined in ISO 8601.

A *complex type* is an element that either contains other elements or has attached attributes. Let's first examine an element with attached attributes and then a more complex element that contains child elements. Here is a definition for a book element that has two attributes called "title" and "pages":

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:attribute name="title" type="xsd:string" />
    <xsd:attribute name="pages" type="xsd:int" />
  </xsd:complexType>
</xsd:element>
```

An XML instance of the book element would look like this:

```
<book title = "More Java Pitfalls" pages="453" />
```

Now let's look at how we define a "product" element with both attributes and child elements. The product element will have three attributes: id, title, and price. It will also have two child elements: description and categories. The categories child element is mandatory and repeatable, while the description child element will be optional:

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string"
minOccurs="0" maxOccurs = "1" />
      <xsd:element name="category" type="xsd:string"
minOccurs = "1" maxOccurs = "unbounded" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" />
    <xsd:attribute name="title" type="xsd:string" />
    <xsd:attribute name="price" type="xsd:decimal" />
  </xsd:complexType>
</xsd:element>
```

Here is an XML instance of the product element defined previously:

```
<product id="P01" title="Wonder Teddy" price="49.99">
  <description>
    The best selling teddy bear of the year.
  </description>
  <category> toys </category>
  <category> stuffed animals </category>
</product>
```

An alternate version of the product element could look like this:

```
<product id="P02" title="RC Racer" price="89.99">
  <category> toys </category>
  <category> electronic </category>
  <category> radio-controlled </category>
</product>
```

Schema definitions can be very complex and require some expertise to construct. Some organizations have chosen to ignore validation or hardwire it into the software. The next section examines this issue.

Is Validation Worth the Trouble?

Anyone who has worked with validation tools knows that developers are at the mercy of the maturity of the tools and specifications they implement. Validation, and the tool support for it, is still evolving. Until the schema languages mature, validation will be a frustrating process that requires testing with multiple tools. You should not rely on the results of just one tool because it may not have implemented the specification correctly or could be buggy. Fortunately, the tool support for schema validation has been steadily improving and is now capable of validating even complex schemas.

Even though it may involve significant testing and the use of multiple tools, *validation is a critical component of your data management process.* Validation is

critical because XML, by its nature, is intended to be shared and processed by a large number and variety of applications. Second, a source document, if not used in its entirety, may be broken up into XML fragments and parts reused. Therefore, the cost of errors in XML must be multiplied across all the programs and partners that rely on that data. As mining tools proliferate, the multiplication factor increases accordingly.

MAXIM

Every XML instance should be validated during creation to ensure the accuracy of all data values in order to guarantee data interoperability.

The chief difficulties with validation stem from the additional complexity of new features introduced with XML Schema: data types, namespace support, and type inheritance. A robust data-typing facility, similar to that found in programming languages, is not part of XML syntax and is therefore layered on top of it. Strong data typing is key to ensuring consistent interpretation of XML data values across diverse programming languages and hardware. Namespace support provides the ability to create XML instances that combine elements and attributes from different markup languages. This allows you to reuse elements from other markup languages instead of reinventing the wheel for identical concepts. Thus, namespace support eases software interoperability by reducing the number of unique vocabularies applications must be aware of. Type inheritance is the most complex new feature in XML Schema and is also borrowed from object-oriented programming. This feature has come under fire for being overly complex and poorly implemented; therefore, it should be avoided until the next version of XML Schema.

As stated previously, namespace support is a key benefit of XML Schema. Let's examine namespaces in more detail and see how they are implemented.

What Are XML Namespaces?

Namespaces are a simple mechanism for creating globally unique names for the elements and attributes of your markup language. This is important for two reasons: to deconflict the meaning of identical names in different markup languages and to allow different markup languages to be mixed together without ambiguity. Unfortunately, namespaces were not fully compatible with DTDs, and therefore their adoption has been slow. The current markup definition languages, like XML Schema, fully support namespaces.

MAXIM

All new markup languages should declare one or more namespaces.

Other Schema-Related Efforts

Two efforts that extend schemas are worth mentioning: the Schema Adjunct Framework and the Post Schema Validation Infoset (PSVI). The Schema Adjunct Framework is a small markup language to associate new domain-specific information to specific elements or attributes in the schema. For example, you could associate a set of database mappings to a schema. Schema Adjunct Framework is still experimental and not a W3C Recommendation.

The PSVI defines a standard set of information classes that an application can retrieve after an instance document has been validated against a schema. For example, an application can retrieve the declared data types of elements and attributes present in an instance document. Here are some of the key PSVI information classes: element and attribute type information, validation context, validity of elements and attributes, identity table, and document information.

Namespaces are implemented by requiring every XML name to consist of two parts: a prefix and a local part. Here is an example of a fully qualified element name:

```
<xsd:integer>
```

The local part is the identifier for the meta data (in the preceding example, the local part is “integer”), and the prefix is an abbreviation for the actual namespace in the namespace declaration. The actual namespace is a unique Uniform Resource Identifier (URI; see sidebar). Here is a sample namespace declaration:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The preceding example declares a namespace for all the XML Schema elements to be used in a schema document. It defines the prefix “xsd” to stand for the namespace “http://www.w3.org/2001/XMLSchema”. It is important to understand that the prefix is not the namespace. The prefix can change from one instance document to another. The prefix is merely an abbreviation for the namespace, which is the URI. To specify the namespace of the new elements you are defining, you use the `targetNamespace` attribute:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.mycompany.com/markup">
```

There are two ways to apply a namespace to a document: attach the prefix to each element and attribute in the document or declare a default namespace for the document. A default namespace is declared by eliminating the prefix from the declaration:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title> Default namespace Test </title> </head>
<body> Go Semantic Web!! </body>
</html>
```

Here is a text representation of what the preceding document is internally translated to by a conforming XML processor (note that the use of braces to offset the namespace is an artifice to clearly demarcate the namespace from the local part):

```
<{http://www.w3.org/1999/xhtml}html>
<{http://www.w3.org/1999/xhtml}head>
<{http://www.w3.org/1999/xhtml}title> Default namespace Test
</{http://www.w3.org/1999/xhtml}title> </head>
<{http://www.w3.org/1999/xhtml}body> Go Semantic Web!!
</{http://www.w3.org/1999/xhtml}body>
</{http://www.w3.org/1999/xhtml}html>
```

This processing occurs during parsing by an application. *Parsing* is the dissection of a block of text into discernible words (also known as tokens). There are three common ways to parse an XML document: by using the Simple API for XML (SAX), by building a Document Object Model (DOM), and by employing a new technique called *pull parsing*. SAX is a style of parsing called *event-based parsing* where each information class in the instance document generates a corresponding event in the parser as the document is traversed. SAX parsers are useful for parsing very large XML documents or in low-memory environments. Building a DOM is the most common approach to parsing an XML document and is discussed in detail in the next section. Pull parsing is a new technique that aims for both low-memory consumption and high performance. It is especially well suited for parsing XML Web services (see Chapter 4 for details on Web services).

What Is a URI?

A Uniform Resource Identifier (URI) is a standard syntax for strings that identify a resource. Informally, URI is a generic term for *addresses* and *names* of objects (or resources) on the World Wide Web. A *resource* is any physical or abstract thing that has an identity.

There are two types of URIs: Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). A URL identifies a resource by how it is accessed; for example, “<http://www.example.com/stuff/index.html>” identifies an HTML page on a server with a Domain Name System (DNS) name of `www.example.com` and accessed via the Hypertext Transfer Protocol (used by Web servers on standard port 80). A URN creates a unique and persistent name for a resource either in the “urn” namespace or another registered namespace. A URN namespace dictates the syntax for the URN identifier.

Pull parsing is also an event-based parsing technique; however, the events are read by the application (pulled) and not automatically triggered as in SAX. Parsers using this technique are still experimental. The majority of applications use the DOM approach to parse XML, discussed next.

What Is the Document Object Model (DOM)?

The *Document Object Model (DOM)* is a language-neutral data model and application programming interface (API) for programmatic access and manipulation of XML and HTML. Unlike XML instances and XML schemas, which reside in files on disk, the DOM is an in-memory representation of a document. The need for this arose from differences between the way Internet Explorer (IE) and Netscape Navigator allowed access and manipulation of HTML documents to support Dynamic HTML (DHTML). IE and Navigator represent the parts of a document with different names, which made cross-browser scripting extremely difficult. Thus, out of the desire for cross-browser scripting came the need for a standard representation for document objects in the browser's memory. The model for this memory representation is object-oriented programming (OOP). So, by turning around the title, we get the definition of a DOM: a data model, using objects, to represent an XML or HTML document.

Object-oriented programming introduces two key data modeling concepts that we will introduce here and visit again in our discussion of RDF in Chapter 6: classes and objects. A *class* is a definition or template describing the *characteristics* and *behaviors* of a real-world entity or concept. From this description, an in-memory instance of the class can be constructed, which is called an object. So, an *object* is a specific instance of a class. The key benefit of this approach to modeling program data is that your programming language more closely resembles the problem domain you are solving. Real-world entities have characteristics and behaviors. Thus, programmers create classes that model real-world entities like "Auto," "Employee," and "Product." Along with a class name, a class has characteristics, known as *data members*, and behaviors, known as *methods*. Figure 3.6 graphically portrays a class and two objects.

The simplest way to think about a DOM is as a set of classes that allow you to create a tree of objects in memory that represent a manipulable version of an XML or HTML document. There are two ways to access this tree of objects: a generic way and a specific way. The generic way (see Figure 3.7) shows all parts of the document as objects of the same class, called Node. The generic DOM representation is often called a "flattened view" because it does not use class inheritance. Class inheritance is where a child class inherits characteristics and behaviors from a parent class just like in biological inheritance.

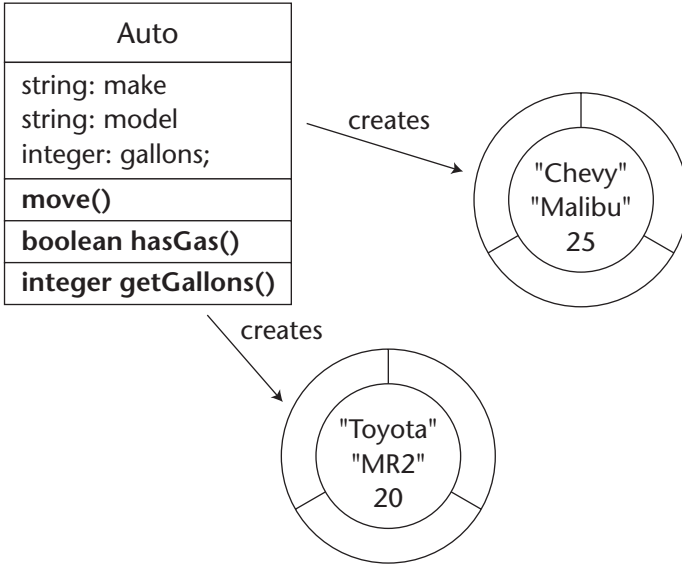


Figure 3.6 Class and objects.

The DOM in Figure 3.7 can also be accessed using specific subclasses of `Node` for each major part of the document like `Document`, `DocumentFragment`, `Element`, `Attr` (for attribute), `Text`, and `Comment`. This more object-oriented tree is displayed in Figure 3.8.

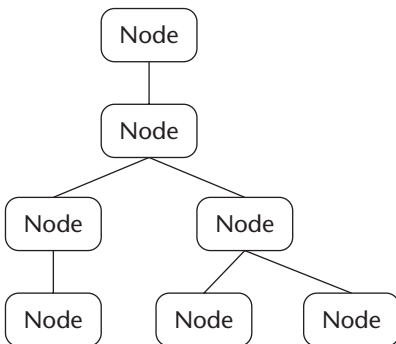


Figure 3.7 A DOM as a tree of nodes.

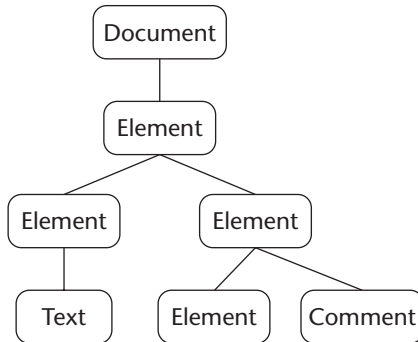


Figure 3.8 A DOM as a tree of subclasses.

The DOM has steadily evolved by increasing the detail of the representation, increasing the scope of the representation, and adding new manipulation methods. This is accomplished by dividing the DOM into conformance levels, where each new level adds to the feature set. There are currently three DOM levels:

DOM Level 1. This set of classes represents XML 1.0 and HTML 4.0 documents.

DOM Level 2. This extends Level 1 to add support for namespaces; cascading style sheets, level 2 (CSS2); alternate views; user interface events; and enhanced tree manipulation via interfaces for traversal and ranges. Cascading style sheets can be embedded in HTML or XML documents in the `<style>` element and provide a method of attaching styles to selected elements in the document. Alternate views allow alternate perspectives of a document like a new DOM after a style sheet has been applied. User interface events are events triggered by a user, such as mouse events and key events, or triggered by other software, such as mutation events and HTML events (load, unload, submit, etc.). Traversals add new methods of visiting nodes in a tree—specifically, `NodeIterator` and `TreeWalker`—that correspond to traversing the flattened view and traversing the hierarchical view (as diagrammed in Figures 3.7 and 3.8). A range allows a selection of nodes between two boundary points.

DOM Level 3. This extends Level 2 by adding support for mixed vocabularies (different namespaces), XPath expressions (XPath is discussed in detail in Chapter 6), load and save methods, and a representation of abstract schemas (includes both DTD and XML Schema). XPath is a language to select a set of nodes within a document. Load and save methods specify a standard way to load an XML document into a DOM and a way to save a DOM into an XML document. Abstract schemas provide classes to represent DTDs and schemas and operations on the schemas.

In summary, the Document Object Model is an in-memory representation of an XML or HTML document and methods to manipulate it. DOMs can be loaded from XML documents, saved to XML documents, or dynamically generated by a program. The DOM has provided a standard set of classes and APIs for browsers and programming languages to represent XML and HTML. The DOM is represented as a set of interfaces with specific language bindings to those interfaces.

Impact of XML on Enterprise IT

XML is pervading all areas of the enterprise, from the IT department to the intranet, extranet, Web sites, and databases. The adoption of XML technology has moved well beyond early adopters into mainstream use and has become integrated with the majority of commercial products on the market, either as a primary or enabling technology. This section examines the current and future impact of XML in 10 specific areas:

Data exchange and interoperability. XML has become the universal syntax for exchanging data between organizations. By agreeing on a standard schema, organization can produce these text documents that can be validated, transmitted, and parsed by any application regardless of hardware or operating system. The government has become a major adopter of XML and is moving all reporting requirements to XML. Companies report financial information via XML, and local governments report regulatory information. XML has been called the next Electronic Data Interchange (EDI) system, which formerly was extremely costly, was cumbersome, and used binary encoding. The reasons for widespread adoption in this area are the same reasons for XML success (listed earlier in this chapter). Easy data exchange is the enabling technology behind the next two areas: ebusiness and Enterprise Application Integration.

Ebusiness. Business-to-business (B2B) transactions have been revolutionized through XML. B2B revolves around the exchange of business messages to conduct business transactions. There are dozens of commercial products supporting numerous business vocabularies developed by RosettaNet, OASIS, and other organizations. Case studies and success stories abound from top companies like Coca-Cola, IBM, Cardinal Health, and Fannie Mae. Web services and Web service registries are discussed in Chapter 4 and will increase this trend by making it even easier to deploy such solutions. IBM's Chief Information Officer, Phil Thompson, recently stated in an interview on CNET, "We have \$27 billion of e-commerce floating through our systems at an operating cost point that is giving us leverage for enhanced profitability."

Enterprise Application Integration (EAI). Enterprise Application Integration is the assembling of legacy applications, databases, and systems to work together to support integrated Web views, e-commerce, and Enterprise Resource Planning (ERP). The Open Applications Group (www.openapplications.org) is a nonprofit consortium of companies to define standards for application integration. It currently boasts over 250 live sites and more than 100 vendors (including SAP, PeopleSoft, and Oracle) supporting the Open Applications Group Integration Specification (OAGIS) in their products. David Chappell writes, “EAI has proven to be the killer app for Web services.”²

Enterprise IT architectures. The impact of XML on IT architectures has grown increasingly important as a bridge between the Java 2 Enterprise Edition (J2EE) platform and Microsoft’s .NET platform. Large companies are implementing both architectures and turning to XML Web services to integrate them. Additionally, XML is influencing development on every tier of the N-tier network. On the client tier, XML is transformed via XSLT to multiple presentation languages like Scalable Vector Graphics (SVG). SVG is discussed in Chapter 6. On the Web tier, XML is used primarily as the integration format of choice and merged in middleware. Additionally, XML is used to configure and deploy applications on this tier like Java Server Pages (JSP) and Active Server Pages (ASP). In the back-end tier, XML is being stored and queried in relational databases and native XML databases. A more detailed discussion of this is provided later in this section.

Content Management Systems (CMS). CMS is a Web-based system to manage the production and distribution of content to intranet and Internet sites. XML technologies are central to these systems in order to separate raw content from its presentation. Content can be transformed on the fly via the Extensible Stylesheet Language Transformation (XSLT) to browsers or wireless clients. XSLT is discussed in Chapter 6. The ability to tailor content to user groups on the fly will continue to drive the use of XML for CMS systems.

Knowledge management and e-learning. Knowledge management involves the capturing, cataloging, and dissemination of corporate knowledge on intranets. In essence, this treats corporate knowledge as an asset. Electronic learning (e-learning) is part of the knowledge acquisition for employees through online training. Current incarnations of knowledge management systems are intranet-based content management systems (discussed previously) and Web logs. XML is driving the future of knowledge management

²David Chappell, “Who Cares about UDDI?”, available at http://www.chappellassoc.com/articles/article_who_cares_UDDI.html.

in terms of knowledge representation (RDF is discussed in Chapter 5), taxonomies (discussed in Chapter 7), and ontologies (discussed in Chapter 8). XML is fostering e-learning with standard formats like the Instructional Management System (IMS) XML standards (at www.imsproject.org).

Portals and data integration. A portal is a customizable, multipaned view tailored to support a specific community of users. XML is supported via standard transformation portlets that use XSLT to generate specific presentations of content (as discussed previously under *Content Management Systems*), syndication of content, and the integration of Web services. A portlet is a dynamically pluggable application that generates content for one pane (or subwindow) in a portal. Syndication is the reuse of content from another site. The most popular format for syndication is an XML-based format called the Resource Description Framework Site Summary (RSS). RDF is discussed in Chapter 5. The integration of Web services into portals is still in its early stages but will enhance portals as the focal point for enterprise data integration. All the major portal vendors are integrating Web services into their portal products.

Customer relationship management (CRM). CRM systems enable an organization's sales and marketing staff to understand, track, inform, and service their customers. CRM involves many of the other systems we have discussed here, such as portals, content management systems, data integration, and databases (see next item), where XML is playing a major role. XML is becoming the glue to tie all these systems together to enable the sales force or customers (directly) to access information when they want and wherever they are (including wireless).

Databases and data mining. XML has had a greater effect on relational database management systems (DBMS) than object-oriented programming (which created a new category of database called object-oriented database management systems, or OODBMS). XML has even spawned a new category of databases called native XML databases exclusively for the storage and retrieval of XML. All the major database vendors have responded to this challenge by supporting XML translation between relational tables and XML schemas. Additionally, all of the database vendors are further integrating XML into their systems as a native data type. This trend toward storing and retrieving XML will accelerate with the completion of the W3C XQuery specification. We discuss XQuery in Chapter 6.

Collaboration technologies and peer-to-peer (P2P). Collaboration technologies allow individuals to interact and participate in joint activities from disparate locations over computer networks. P2P is a specific decentralized collaboration protocol. XML is being used for collaboration at the protocol

level, for supporting interoperable tools, configuring the collaboration experience, and capturing shared content. XML is the underpinning of the open source JXTA project (www.jxta.org).

XML is positively affecting every corner of the enterprise. This impact has been so extensive that it can be considered a data revolution. This revolution of data description, sharing, and processing is fertile ground to move beyond a simplistic view of meta data. The next section examines why meta data is not enough and how it will evolve in the Semantic Web.

Why Meta Data Is Not Enough

XML meta data is a form of description. It describes the purpose or meaning of raw data values via a text format to more easily enable exchange, interoperability, and application independence. As description, the general rule applies that “more is better.” Meta data increases the fidelity and granularity of our data. The way to think about the current state of meta data is that we attach words (or labels) to our data values to describe it. How could we attach sentences? What about paragraphs? While the approach toward meta data evolution will not follow natural language description, it is a good analogy for the inadequacy of words alone. The motivation for providing richer data description is to move data processing from being tediously preplanned and mechanistic to dynamic, just-in-time, and adaptive.

For example, you may be enabling your systems to respond in real time to a location-aware cell phone customer who is walking by one of your store outlets. If your system can match consumers’ needs or past buying habits to current sale merchandise, you increase revenue. Additionally, your computers should be able to support that sale with just-in-time inventory by automating your supply chain with your partners. Finally, after the sale, your systems should perform rich customer relationship management by allowing transparency of your operations in fulfilling the sale and the ability to anticipate the needs of your customers by understanding their life and needs. The general rule is this: The more computers understand, the more effectively they can handle complex tasks.

We have not yet invented all the ways a semantically aware computing system can drive new business and decrease your operation costs. But to get there, we must push beyond simple meta data modeling to knowledge modeling and standard knowledge processing. Here are three emerging steps beyond simple meta data: semantic levels, rule languages, and inference engines.

Semantic Levels

Figure 3.9 shows the evolution of data fidelity required for semantically aware applications. Instead of just meta data, we will have an information stack composed of semantic levels. We are currently at Level 1 with XML Schema, which is represented as modeling the properties of our data classes. We are capturing and processing meta data about isolated data classes like purchase orders, products, employees, and customers. On the left side of the diagram we associate a simple physical metaphor to the state of each level. Level 1 is analogous to describing singular concepts or objects.

In Level 2, we will move beyond data modeling (simple meta data properties) to knowledge modeling. This is discussed in detail in Chapter 5 on the Resource Description Framework (RDF) and Chapter 7 on taxonomies. Knowledge modeling enables us to model statements both about the relationships between Level 1 objects and about how those objects operate. This is diagrammed as connections between our objects in Figure 3.9.

Beyond the knowledge statements of Level 2 are the superstructures or “closed-world modeling” of Level 3. The technology that implements these sophisticated models of systems is called ontologies and is discussed in Chapter 8.

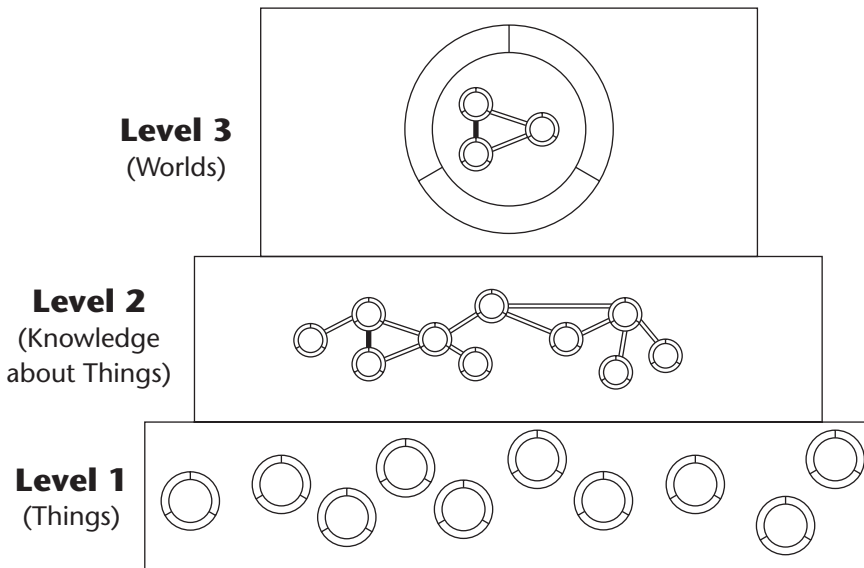


Figure 3.9 Evolution in data fidelity.

How can we be sure this evolution will happen and will deliver a return on investment? The evolution of data fidelity and realism has already occurred in many vertical industries to include video games, architecture (computer-aided drafting), and simulations (weather, military, and so on). As an analogy to the effects of realism, a simple test would be to attempt to convince a teenager to play a 1970s arcade-style game like Asteroids instead of any of the current three-dimensional first-person shooter games. Figure 3.10 displays the fidelity difference between the original action arcade game SpaceWar and a high-fidelity combat game called Halo. My 12-year-old son will eagerly discuss the advanced physics of the latest game on the market and why it blows away the competition. How do we apply this to business? High-fidelity, closed-world models allow you to know your customer better, respond faster, rapidly set up new business partners, improve efficiencies, and reduce operation costs. For dimensions like responsiveness, just-in-time, and tailored, which are matters of degree, moving beyond simple meta data will produce the same orders of magnitude improvement as demonstrated in Figure 3.10.

Rules and Logic

The semantic levels of information provide the input for software systems. The operations that a software system uses to manipulate the semantic information will be standardized into one or more rule languages. In general, a rule specifies an action if certain conditions are met. The general form is this: if (x) then y. Current efforts on rule languages are discussed in Chapters 5 and 8.

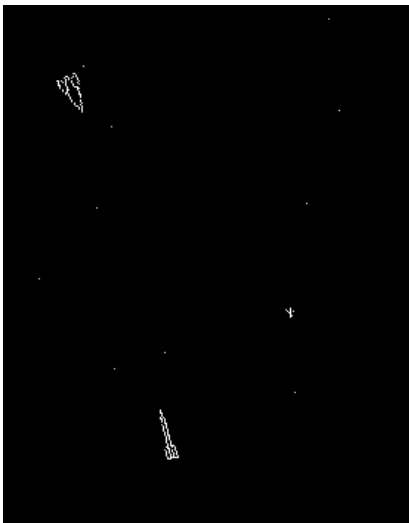


Figure 3.10 Data fidelity evolution in video games.

SpaceWar by Stern, from the Spacewar emulator at the MIT Media Lab

Inference Engines

Applying rules and logic to our semantic data requires standard, embeddable inference engines. These programs will execute a set of rules on a specific instance of data using an ontology. An early example of these types of inferring engines is the open source software Closed World Machine (CWM). CWM is an inference engine that allows you to load ontologies or closed worlds (Semantic Level 3), then it executes a rule language on that world.

So, meta data is a starting point for semantic representation and processing. The rise of meta data is related to the ability to reuse meta data between organizations and systems. XML provides the best universal syntax to do that. With XML, everyone is glimpsing the power of meta data and the limitations of simple meta data. The following chapters examine how we move beyond meta data toward knowledge.

Summary

This chapter provided an in-depth understanding of XML and its impact on the enterprise. The discussion was broken down into four major sections: XML success factors, the mechanics of XML, the impact of XML, and why simple data modeling is not enough.

There are four primary reasons for XML's success:

XML creates application-independent documents and data. XML can be inspected by humans and processed by any application.

It has a standard syntax for meta data. XML provides an effective approach to describe the structure and purpose of data.

It has a standard structure for both documents and data. XML organizes data into a hierarchy.

XML is not a new technology (not a 1.0 release). XML is a subset of SGML, which has been around more than 30 years.

In understanding the mechanics of XML, we examined what markup is, the syntax of tags, and how start, end, and empty tags are used. We continued to explore the mechanics of XML by learning the difference between well-formed and valid documents, how we define the legal elements and attributes using XML Schema, how to use namespaces to create unique element and attribute names, and how applications and browsers represent documents internally using the Document Object Model. After understanding XML, we turned to its impact on the enterprise. XML has had considerable impact on 10 areas: data

exchange, ebusiness, EAI, IT architectures, CMS, knowledge management, portals, CRM, databases, and collaboration. XML's influence will increase dramatically over the next 10 years with the advent of the Semantic Web.

Lastly, we turned a critical eye on the current state of XML meta data and why it is not enough to fulfill the goals of the Semantic Web. The evolution of meta data will expand into three levels: modeling of things, modeling of knowledge about things, and, finally, modeling "closed worlds." In addition to modeling knowledge and worlds, we will expand to model the rules and axioms of logic in order for computers to automatically use and manipulate those worlds on our behalf. Finally, to apply those rules, standard inference engines, like CWM, will be created and embedded into many of the current IT applications.

In conclusion, XML is a strong foundation for the Semantic Web. Its next significant stage of development is the advent of Web services, discussed in the next chapter.

